A Study of the Influence of the POWER5 Dynamic Resource Balancing (DRB) on Optimal Hardware Thread Priorities

Princess C. Trillo, Undergraduate Student,
Mitesh R. Meswani, Ph.D. Student,
Patricia J. Teller, Ph.D., Professor, and
Sarala Arunagiri, Ph.D., Research Specialist
Department of Computer Science, University of Texas at El Paso

Abstract

Simultaneous Multithreading, often abbreviated SMT, is a technique for improving the overall efficiency of superscalar processors with hardware multithreading. SMT permits a processor to concurrently execute multiple independent instruction streams every clock cycle, potentially improving processor throughput. However, this can introduce contention for shared resources amongst threads running concurrently in SMT mode. In order to enable the programmer to control the ratio in which resources are shared, the IBM POWER5 processor allows prioritization of one thread over another. The processor also implements Dynamic Resource Balancing (DRB) hardware, which throttles back a thread that monopolizes architectural resources by reducing its thread priority. Unlike thread priorities, the DRB is not tunable by software. In this paper, the hardware thread priorities that give best processor throughput are referred to as optimal hardware thread priorities.

The research described in this paper answers the following question: Does the POWER5's DRB influence the identification of optimal hardware thread priorities for a given pair of threads running concurrently in SMT mode, i.e., a co-schedule? To answer this question we used a POWER5 simulator and compared cycles per instruction (CPI) with DRB enabled and DRB disabled while simulating application runs for application pairs composed of SPEC CPU2000 and STREAM benchmarks. Our results show that (1) there was less than 1% difference between the CPIs of the threads of all co-schedules except for co-schedules executing a SPEC floating-point intensive benchmark and a SPEC integer-intensive benchmark; (2) whether DRB is enabled or disabled, approximately 40% of co-schedules do not experience best performance with equal priorities; and (3) approximately 69% of the co-schedules experienced best performance at the same priorities with DRB enabled and DRB disabled. Thus, the enabling or disabling of the POWER5's DRB does not have a significant impact on the identification of a co-schedule's optimal thread priorities.

1. Introduction and Background

The disparity between the speeds of memory and processors has led to reduced processor utilization and throughput. To address this problem, computer architects have implemented techniques within the processor pipeline to take advantage of the inherent parallelism of applications, called instruction-level parallelism (ILP). However, processors continued to remain underutilized [1].

Simultaneous multithreading, often abbreviated SMT, is a technique used to improve processor utilization. A processor with SMT interleaves the execution of multiple instruction streams in the pipeline every clock cycle. This interleaving allows resources that are left idle by one thread to be used by another, thus, improving throughput by taking advantage of parallelism amongst multiple threads, called thread-level parallelism (TLP). A superscalar processor with SMT can take advantage of thread-level and instruction-level parallelism to improve processor utilization and, thus, throughput. IBM [2] and Intel [3] have implemented SMT on superscalar processors.

Hardware threads of an SMT processor share most of the processor's resources at cycle-level granularity. In general, concurrently executing applications (hereinafter called threads) that do not compete for the same classes of resources, such as memory, increase processor throughput. However, throughput drops when the threads compete aggressively for the same classes of resources. The prediction of contention for shared resources depends on the characteristics of the concurrently executing threads.

To address the issue of reduced throughput due to resource contention, some SMT processors allow enabling and disabling of SMT mode [2, 3, 17], and prioritization of one thread over the other [3, 17].

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JAN 2009		**00-00-2009 to 00-00-2009**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
A Study of the Influence of the POWER5 Dynamic Resource Balancing (DRB) on Optimal Hardware Thread Priorities	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Texas at El Paso,Department of Computer Science,El Paso,TX, 79968**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Live-Virtual Constructive Conference, 12-15 Jan 2009, El Paso, TX

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**8**	

The IBM POWER5 [2] implements SMT with two hardware threads per core. To control resource contention by the hardware threads, the POWER5 allows software to enable and disable SMT mode and assign thread priorities that allow prioritization of one thread over another by increasing the proportion of decode cycles allocated to the higher-priority thread. The broader research topic, of which this project is a part, seeks to develop a methodology to improve the throughput of the IBM POWER5 processor by tuning thread priorities. In particular, our research group is investigating the use of application characteristics of CPU-intensive and memory-intensive workloads to develop a model that can predict "optimal" settings of POWER5 priorities to improve throughput. This body of research was motivated by our initial results [10], which show that SMT processor throughput can be increased by using optimal priority settings instead of the default settings. This study was conducted by simulating application runs using traces from the SPEC CPU2000 [5] and STREAM [6] benchmark suites.

The POWER5 also has built-in hardware, called Dynamic Resource Balancing (DRB) logic, to prevent one thread from monopolizing the processor's microarchitecture resources; the DRB is not tunable by software. With this knowledge, we wanted to investigate whether the results of [10] were influenced in any way by the simulation of the DRB logic. The research described in this paper answers the question: For a given pair of threads is the "optimal" pair of thread priorities dependent on whether DRB is enabled or disabled?

To answer this question, we ran a number of simulations on the IBM performance simulator for Linux on POWER [4] with DRB logic enabled and disabled, and studied the processor performance in terms of cycles per instruction (CPI). For these simulations, we used 11 different combinations of thread priority pairs for 394 pairs of concurrently executing applications (hereinafter called co-schedules). The applications used for this study were SPEC CPU2000 [5] floating-point intensive and integer-intensive benchmarks, as well as memory-intensive benchmarks from the STREAM suite [6].

Our results show that

(1) the differences between the CPIs of the threads of co-schedules, executed with DRB enabled versus DRB disabled, was less than 1%;
(2) whether DRB is enabled or disabled, approximately 40% of co-schedules do not experience best performance with equal priorities; and
(3) approximately 69% of co-schedules experienced best performance at the same priorities with DRB enabled and DRB disabled.

Thus, enabling or disabling of the POWER5's DRB does not have a significant impact on "optimal" thread priorities for co-schedules comprised of floating-point intensive and integer-intensive benchmarks in the SPEC CPU 2000 suite and benchmarks in the STREAM suite.

The rest of the paper is organized as follows. Section 2 discusses related research. The broader project, of which this study is a part, is outlined in Section 3. The microarchitecture details of the POWER5 are described in Section 4. The experimental methodology and the methods used for data collection are explained in Sections 5 and 6, respectively. The simulation results are presented and discussed in Section 7. Finally, Section 8 presents conclusions and future work.

2. Related Research

SMT was first introduced in a simulation of alpha binaries developed by a group of researchers at the University of Washington in 1995 [7]. Subsequently, SMT was introduced in superscalar processors by Intel as Hyper-Threading [2] and by IBM in their POWER5 and POWER6 processors [3, 17].

Intel and IBM SMT processors allow enabling and disabling of SMT mode. Furthermore, the IBM POWER5 and POWER6 processors allow prioritization of one hardware thread over another by using priorities [8] to control the ratio of allocated decode cycles for two threads running concurrently in SMT mode.

The Linux 2.6 [15] and AIX 5L [16] operating systems for POWER5 are SMT-aware [9]. Both operating systems treat each hardware thread as a logical processor and provide run queues for each thread. They also provide SMT-aware load balancing support. AIX and Linux use hardware thread priorities to improve kernel performance. For example, given a co-schedule comprised of two threads that share a lock, they lower the priority of the thread that is busy-waiting on the spinlock, allowing a larger allocation of processor resources to the thread that holds the lock.

According to a study performed at the University of Texas at El Paso [10], SMT processor throughput can be increased by using optimal priority settings instead of the default settings.

3. Project Goals

SMT has the potential to improve processor utilization and, thus, overall processor throughput. However, applications executing concurrently on hardware threads of an SMT processor compete with each other for shared processor resources, such as the cache memory. As a result, concurrently executing applications may interfere with each other's performance. Inter-application interference can have a considerable effect on the overall utilization and, thus, throughput of the processor. Modern operating systems provide specific support for thread scheduling and resource management of SMT processors. However, they do not alleviate performance degradation due to inter-application interference.

The broader research is attempting to develop a methodology to improve the throughput of the IBM POWER5 processor by using application resource-usage characteristics. These characteristics are used to find the "optimal" settings of knobs on SMT processors that improve processor throughput, e.g., hardware thread priorities and the enabling/disabling of SMT mode. We are studying applications that are compute-intensive and memory-intensive; since a processor context switches on I/O activity, we do not study I/O-intensive applications.

The methodology collects processor resource-usage characteristics of applications running in Single-Threaded (ST) mode, i.e., SMT mode disabled. The ST characteristics of the two applications are used to predict settings of hardware thread priorities, explained in Section 4, for improving the application-pair's processor throughput in SMT mode. In order to cover a broad range of co-schedules, we study both homogeneous and heterogeneous workloads. Heterogeneous co-schedules consist of applications that stress different classes of processor resources, whereas homogeneous co-schedules consist of applications that stress the same classes of processor resources. Interference and, thus, performance of heterogeneous co-schedules is dependent on the characteristics of the concurrently executing pair of applications. In contrast, homogeneous co-schedules may produce the worst case scenario for inter-application interference and the associated SMT processor throughput.

A thread that uses more than its fair share of processor resources may prevent its sibling thread from making reasonable progress and, thereby, reduce overall throughput. To address this issue the POWER5 has DRB logic implemented in hardware, described in Section 4, which throttles down a thread that hogs processor resources. The work presented in this paper investigates the influence of the DRB on the "optimal" pair of thread priorities for a given co-schedule.

4. IBM POWER5

The POWER5 [2] chip contains two identical processor cores running at 1.65 GHz; each processor core supports two logical hardware threads. Each processor core has its own 62 KB level-one (L1) instruction cache and a 32 KB L1 data cache. The L1 caches are shared between the two hardware threads of each core. Both cores in the processor share a 1.9 MB unified, in-line, level-two (L2) cache that is fully inclusive of the L1 instruction and data caches. A level-three (L3) cache controller (which provides for an L3 cache directory) is located on-chip; however, the L3 cache, itself, is implemented off-chip. The L3 cache is a 36 MB victim cache, i.e., it stores data and instructions evicted from the L2 cache; thus, the L3 cache is not inclusive of the L2 cache. The L3 cache is shared by both processor cores of the POWER5 chip. All hardware threads of both processor cores access the L2 and L3 caches.

The POWER5 chip supports both SMT and Single-Threaded (ST) execution modes. In SMT mode, there is a program counter (PC) for each hardware thread and instruction-fetch alternates between the two PCs. In ST mode only one PC is used and all instructions are fetched from that thread during every cycle. After a fetch, the instructions are placed in separate instruction buffers for each thread. Based on the priority setting, a certain number of instructions of a hardware thread are fetched from the instruction buffers, and a group is formed. All instructions in a group belong to the same thread. All instructions within a group are decoded in parallel and dispatched. To simplify the logic of tracking instructions through the pipeline, instructions are tracked as a group. At the time of dispatch, control information for each group of instructions is stored in a global completion table (GCT).

In a multi-threaded system, one thread might monopolize the processor resources and block other threads from executing. The POWER5 implements Dynamic Resource Balancing (DRB) logic, which helps concurrently run its two hardware threads effectively on the system. DRB logic monitors L2 miss queues and GCT entries; if a thread exceeds a threshold for any of these resources, the DRB detects this condition and throttles that thread down so that the other thread can make forward progress. The DRB is implemented at the hardware level and is not tunable by software. Depending on the situation, one of three available mechanisms is used to throttle down a

dominating thread: (1) reducing the priority of a thread that exceeds a threshold number of GCT entries; (2) inhibiting instruction decode of a thread that has exceeded a threshold number of outstanding L2 cache misses; and (3) flushing instructions from issue queues of a thread that has long latency instructions.

Adjustable thread priorities [9] on the POWER5 processor allow software to determine the number of decode cycles allocated to each hardware thread; the more decode cycles, the higher the potential of having more execution resources. Eight software-controlled priority levels are supported for each thread in the range 0 to 7 (highest), with the higher priority thread receiving more decode cycles. Three execution modes are supported in the POWER5: (1) hypervisor mode, in which all priority levels can be set; (2) supervisor mode, in which priorities 1 through 6 can be set; and (3) user mode, which is restricted to priorities 2 through 4.

Given that the priority of thread 1 is X and that of thread 2 is Y, the share of decode cycles allocated to thread 1 is given by equation (1); the number of cycles allocated to thread 2 is calculated as: 1 – decode-cycles share of thread 1. The number of decode cycles associated with the possible thread priorities are shown in Table 1. By default, the processor assigns the priority of 4 to both threads.

$$\frac{1}{2^{(|X-Y|+1)}} \qquad (1)$$

Thread X Priority	Thread Y Priority	Priority Difference	Thread X Decode Cycles	Thread Y Decode Cycles
2	7	-5	1/64	63/64
3	7	-4	1/32	31/32
4	7	-3	1/16	15/16
2	4	-2	1/8	7/8
3	4	-1	1/4	3/4
4	4	0	1/2	1/2
4	3	1	3/4	1/4
4	2	2	7/8	1/8
7	4	3	15/16	1/16
7	3	4	31/32	1/32
7	2	5	63/64	1/64

Table 1 Decode Cycles for Thread Priorities

5. Experimental Environment

The research described in this paper determines if the optimal pair of thread priorities is dependent on whether the DRB logic implemented on the POWER5 processor is enabled or disabled. The DRB is implemented at the hardware level and is not tunable by software. Therefore, in order to study the performance of the POWER5 processor without the influence of the DRB logic, we used an instruction trace simulator [11] for the POWER5 that allows enabling and disabling of the DRB. Section 5.1 describes the simulator and its settings and Section 5.2 describes the workload traces studied and the methodology used to capture the traces that were used to drive the simulator.

5.1. Simulator

In 2005 IBM released its IBM performance simulator for Linux on POWER [12], which is "a suite of performance models based on IBM's POWER series of processors." This tool allows users of the Linux operating system running on a POWER processor to examine how an application executes on various IBM POWER processor models so that performance hazards can be identified and prevented on such systems. The simulator requires the use of the Linux operating system on a 64-bit IBM POWER-based computer as well as the IBM vacpp run-time library for Linux [13].

The IBM performance simulator for Linux on POWER requires as input, instruction traces and produces as output, a processor performance report with detailed information on metrics such as CPI, functional unit usage statistics, and instruction histograms for each thread.

The relevant command line arguments required to run the simulator are the following:

- num_inst: the number of architected instructions to run before exiting: set to greater than the combined number of instructions of the two traces
- t0_prio: priority of thread 0
- t1_prio: priority of thread 1

The trace-driven simulator is configured in SMT mode with the desired thread priority combination. The simulator in SMT mode exits and, thus, data collection stops when the processing of one of the concurrently executing traces completes. This is done because the goal of the research is to study the effect of the DRB, which is effective only in SMT mode.

The execution of a trace co-schedule on a single processor of a two-processor POWER5 running the Linux 2.6.17 kernel was simulated. The simulated system has 5 GB main memory and a 70 GB hard disk.

5.2. Workload Traces

The benchmarks used to generate the traces used in this study are described in Sections 5.2.1 and 5.2.2, and instruction tracing is described in Section 5.2.3.

5.2.1. SPEC CPU2000

Benchmarks in the SPEC CPU2000 [5] suite are compute-intensive applications with working sets that fit in main memory. The suite consists of floating-point intensive and integer-intensive applications. Tables 2 and 3 show the SPEC CPU2000 benchmarks that were used in this study.

5.2.2 STREAM2

The STREAM benchmark suite [6] is designed to stress the memory subsystem. The second version of the benchmark, called stream2, measures sustained memory bandwidth at all levels of the memory. The benchmark executes four vector kernels (fill, copy, daxpy, and sum) described in [6]. During execution, the benchmark uses 32 different array sizes proceeding from the minimum size to the maximum size via increments calculated using Equation (2), where *j* ranges from 1 to 32. The amount of work done for any vector length is equal to the maximum array size. Our configuration of stream 2 sets the maximum array size to 120,000,000, which is much larger than the simulated L3 cache. The minimum array size is the default, i.e., 30.

$$Array\ Size = 10^{1.477+(\frac{j-1}{31})*6.6028} \qquad (2)$$

In our study, we use two different versions of stream2 to generate two different traces, one that stresses the on-chip L1 and L2 caches, and one that stresses the off-chip L3 cache and main memory. These versions are called stream2_L2 and stream2_L3, respectively. The way that traces are collected is described in the next section.

Benchmark Name	Description
applu	Parabolic / Elliptic Partial Differential Equations
apsi	Meteorology: Pollutant Distribution
art	Image Recognition / Neural Networks
equake	Seismic Wave Propagation Simulation
fma3d	Finite-element Crash Simulation
galgel	Computational Fluid Dynamics
mesa	3-D Graphics Library
mgrid	Multi-grid Solver: 3D Potential Field
sixtrack	High Energy Nuclear Physics Accelerator Design
wupwise	Physics / Quantum Chromodynamics

Table 2 SPEC CPU2000 Floating-point Benchmarks

5.2.3 Instruction Tracing

The instruction traces used for this experiment were collected using the Linux tool *ITrace* [11] on a preliminary study performed by Meswani and Teller at the University of Texas at El Paso [10].

The instruction traces are captured in a main memory buffer, the size of which is limited to 200 MB per CPU. It was experimentally determined that a ten-second time interval is required to fill the trace buffer and capture the largest possible trace. During this time interval, *ITrace* can capture between 3 and 10 million instructions for the benchmarks described above.

Benchmark Name	Description
crafty	Game Playing: Chess
gap	Group Theory, Interpreter
gcc	C Programming Language Compiler
mcf	Combinatorial Optimization
parser	Word Processing
perlbmk	PERL Programming Language
twolf	Place and Route Simulator
vortex	Object-oriented Database

Table 3 SPEC CPU2000 Integer Benchmarks

6. Data Collection

The performance data presented in this paper was collected from a simulator introduced in Section 5.1. The application groups considered for the study are discussed in Section 6.1. The performance metrics analyzed from the simulation results are presented in Section 6.2.

6.1. Application Co-schedules

In order to examine the performance of the POWER5 processor for homogeneous and heterogeneous workloads, we studied six different co-schedule groups. For homogeneous workloads, we studied three groups: (1) co-schedules comprised of floating-point intensive

applications; (2) co-schedules comprised of integer-intensive applications; and (3) co-schedules comprised of memory-intensive applications. On the other hand, for heterogeneous workloads, we considered: (1) co-schedules comprised of floating-point intensive and integer-intensive applications; (2) co-schedules comprised of floating-point intensive and memory-intensive applications; and (3) co-schedules comprised of integer-intensive and memory-intensive applications.

In our study, a co-schedule refers to an application pair A,B running on hardware threads X,Y, respectively. The hardware threads X,Y can be set at priorities (*i*,*j*), respectively. The threads can be set at one of the following eleven thread priority combinations: (2,4), (2,7), (3,4), (3,7), (4,2), (4,3), (4,4), (4,7), (7,2), (7,3), (7,4). Thus, every co-schedule can be run in 11 different priority setting combinations. Overall, we simulated 394 co-schedules at eleven different priority setting combinations, for a total of 4,334 experiments, once with DRB enabled and once with DRB disabled

6.2. Performance Metric

The cycles per instruction (CPI) metric measured for each experiment executed on the simulator for Linux on POWER was obtained after one of the hardware threads of a co-schedule finished execution. The formula used to calculate the percentage difference of CPI for a co-schedule executed with DRB enabled versus DRB disabled is shown in Equation (3).

$$\frac{100 * (CPIDRBenabled - CPIDRBdisabled)}{CPIDRBdisabled} \quad (3)$$

7. Results

Using the processor usage data reported by the simulation model, we recorded the CPI for each experiment with DRB enabled and DRB disabled. For every co-schedule we calculated the difference in CPI between the experiments with DRB enabled and DRB disabled. For every co-schedule, we recorded the optimal pair of thread priorities with DRB enabled and DRB disabled. We also documented all cases for which the ordering of the pairs of thread priorities did not hold for DRB enabled and DRB disabled.

For tables and graphs in this section, we refer to the group of co-schedules comprised of SPEC CPU2000 floating-point intensive benchmarks as SPECFlt. The group comprised of SPEC CPU2000 integer-intensive benchmarks is referenced as SPECInt. The group of co-schedules comprised of SPEC CPU2000 floating-point intensive and integer-intensive benchmarks is referenced as MixedPairs. For the co-schedules consisting of STREAM memory-intensive benchmarks and SPEC CPU2000 floating-point intensive benchmarks, the group is called Stream_SPECFP. The group comprised of co-schedules of the STREAM memory-intensive benchmarks and SPEC CPU2000 integer-intensive benchmarks is referred to as Stream_SPECINT. The last group of co-schedules, comprised of STREAM memory-intensive benchmarks, is referenced as Stream_Stream.

The main contribution of this study and other observations made as a result of this study are presented in Section 7.1 and 7.2, respectively.

7.1 Main Contribution

Table 4 presents the absolute average percentage CPI difference between experiments completed with DRB enabled versus DRB disabled for a given application group. As can be seen from this table, the average percentage difference in CPI amongst all groups is less than 1%. This shows that results from our previous study [10] were essentially unaffected by the DRB hardware for the set of applications and size of traces simulated. Table 4 also implies that the DRB logic does not influence the identification of the optimal pair of thread priorities.

7.2 Other Observations

Figure 1 shows the percentage of co-schedules for a given application group that did not experience best performance with equal thread priorities whether DRB was enabled or disabled. As can be seen from this graph, approximately 42% of co-schedules across all groups did not experience best performance at equal priorities. The top bar of each group indicates the percentage of co-schedules for that group that experienced the best performance at unequal priorities when DRB was enabled. The bottom bar shows the percentage of co-schedules that yielded the best performance at unequal priorities when DRB was disabled. Independent of the DRB logic being enabled or disabled, for 42% of the experiments, the best performance from the processor was obtained when the thread priorities for a co-schedule were not equal.

Next, we analyze if the best priority settings for a co-schedule changes if DRB is enabled or disabled. As shown in Figure 2, the average percentage of co-schedules that experienced the best performance at equal priorities with DRB enabled and disabled across groups is 69%.

Next, we analyze if the ordering of priorities by throughput, going from best case to worst case, for an application pair changed if DRB was enabled or disabled. For 78% of all co-schedules, the ordering of thread priorities changes if DRB is enabled or disabled. In other words, the enabling and disabling of DRB affects the ordering of priorities for a given co-schedule. However, we also observed that for 63% of these co-schedules, the best case priority settings remains the same for DRB enabled and DRB disabled. These results are shown in Figure 3.

Our experiments show that the enabling or disabling of DRB on the POWER5 processor does not have a significant effect on determining the optimal pair of thread priorities for the applications used in this study.

Group Name	Average % delta in CPI for DRB Enabled and Disabled
SPECFlt	0.922%
SPECInt	0.696%
MixedPairs	0.950%
Stream_SPECFP	0.851%
Stream_SPECINT	0.851%
Stream_Stream	0.673%

Table 4 Average Percentage Difference in CPI for DRB Enabled and Disabled

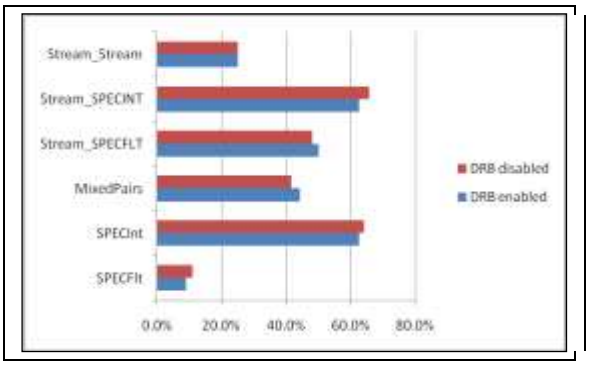

Figure 1. Application Pairs for which Best Thread Priorities were not Default (4, 4)

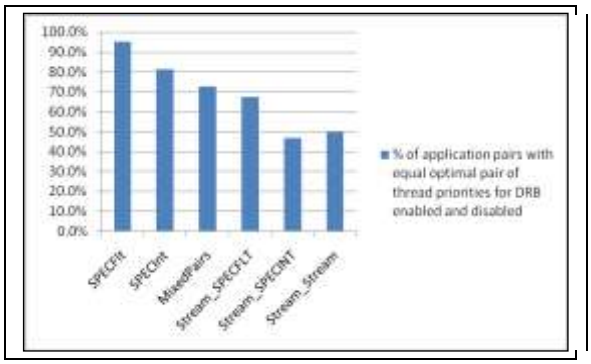

Figure 2. Application Pairs with the Same Best Thread Priorities with DRB Enabled or Disabled

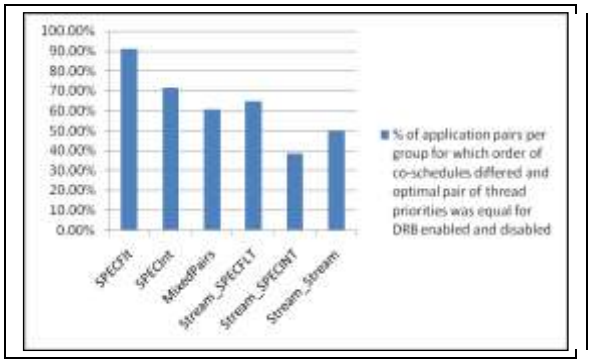

Figure 3. Application Pairs with the Same Best Thread Priorities but Different Orderings of Remaining Priorities with DRB Enabled and Disabled

8. Conclusions and Future Work

The research described in this paper shows that the enabling or disabling of the DRB logic on the POWER5 processor does not have a significant impact on identification of optimal pairs of thread priorities for co-schedules comprised of floating-point intensive and integer-intensive benchmarks in the SPEC CPU2000 suite and benchmarks in the STREAM memory-intensive suite. The results of this work are based on application traces. If it were possible, we would verify the results of this study using a processor that allows the enabling and disabling of the DRB and complete applications.

Acknowledgements

We would like to acknowledge that this work was supported by the Spreading High Performance computing Participation in undergraduate Education and Research grant of the National Science Foundation and the Army High Performance Computing Research Center (AHPCRC) grant W11NF-07-2-2007.

9. References

[1] J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufman, CA, 2003.

[2] R. Kalla, B. Sinahroy, and J. M. Tendler, "IBM POWER5 Chip: a Dual-Core Multithreaded Processor," *IEEE Micro*, March 2004, 24(2):40-47.

[3] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller, and M. Upton, "Hyper-Threading Technology Architecture and Microarchitecture," *Intel Technology Journal*, February 2002, 3(1):4-15.

[4] "alphaWorks: IBM Performance Simulator for Linux on POWER: Overview," http://www.alphaworks.ibm.com/tech/simppc, accessed October 30, 2008.

[5] SPEC - Standard Performance Evaluation Corporation, http://www.spec.org/cpu2000, accessed October 30, 2008.

[6] Memory Bandwidth: Stream Benchmark Performance Results, http://www.cs.virginia.edu/stream/, accessed October 20, 2008.

[7] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous Multithreading: Maximizing On-chip Parallelism," *Proceedings of the 22th International Symposium on Computer Architecture (ISCA '95)*, IEEE Computer Society, June 1995, pp. 392-403.

[8] H. M. Mathis, A. E. Mericas, J. D. McCalpin, R. J. Eickemeyer, and S. R. Kunkel, "Characterization of Simultaneous Multithreading (SMT) Efficiency in POWER5," *IBM Journal of Research and Development*, July 2005, 49(4/5):555-564.

[9] Advanced POWER Virtualization on IBM eServer p5 Servers: Architecture and Performance Considerations, SG245768, http://www.redbooks.ibm.com/abstracts/sg245768.

[10] M. R. Meswani and P. J. Teller, "Evaluating the Performance Impact of Hardware Thread Priorities in Simultaneous Multithreaded Processors using SPEC CPU2000," *Proceedings of the 2nd International Workshop on Operating Systems Interference in High Performance Applications (OSIHPA)*, Seattle, WA, September 2006.

[11] "ITrace for Linux/PPC," http://perfinsp.sourceforge.net/itrace_ppc.html, accessed October 30, 2008.

[12] "alphaWorks : IBM Performance Simulator for Linux on Power: Overview, " http://www.alphaworks.ibm.com/tech/simppc, accessed October 30, 2008.

[13] "alphaWorks : IBM Performance Simulator for Linux on Power: Download, " http://www.alphaworks.ibm.com/tech/simppc/download, accessed October 30, 2008.

[14] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden, "IBM POWER6 Microarchitecture," *IBM Journal of Research and Development*, November 2007, 51(6):639-662.

[15] "The Linux Kernel Archives," http://www.kernel.org/, accessed October 30, 2008.

[16] "IBM Power Systems software – AIX," http://www-03.ibm.com/systems/power/software/aix/index.html, accessed October 30, 2008.